Ordered Matrix Inversion

John Gipson

December 17, 1998

Abstract

I derive the equations for two forms of fast-matrix inversion. I start by describing arc-parameter elimination, and derive the explicit equation for the solution vector, and for the covariance matrix. I generalize these results to the solution of "time-tagged" sparse normal matrces. These are normal equations where for a large number of parameters data from only a small interval contributes. Inverting the normal equations in time order results in a substantial reduction in the time required to solve the normal equations. I refer to this technique as "Ordered Matrix Inversion". I compare this technique to the "B3D" algorithm derived by L. Petrov.

1 Introduction

Many problems deal with finding the solution to the matrix equations

$$NA = B$$

where N is an $n \times n$ matrix, and A and B are n dimensional column vectors. In experimental sciences these equations show up as the normal equations, and in what follows this is how I will refer to them. If both the solution vector A and the covariance matrix $N^{-1}$are required, a straightforward solution of the normal equations requires approximately $n^3/2$ multiplications. If only the solution vector A is required, the number of operations is reduced by a factor of 3 to $n^3/6$. Even with this reduction in computational cost, a straightforward solution is impractical for large matrices. Because of this one must try to find various tricks to make the inversion faster. If the matrix is sparse there exist canned routines which will perform the inversion for an arbitrary matrix. The disadvantage of this approach is that since these are general purpose routines, they may not be the most efficient. If you have some knowledge about the structure of the normal matrix you may be able to invert it more efficiently. In this note I describe two techniques for inverting sparse matrices which appear in geophysics.

In section 2 I describe arc-parameter elimination, which has been described elsewhere in the literature. I introduce it here in order to establish notation, and becaue it is a percursor to a more general technique. Arc-parameter elimination arises in doing a combined solution from a series of measurements. Some of the estimated parameters are common to many or all of the experiments, and are called "global" parameters. For example, these might be station positions. The remaining parameters appear in only a single experiment or arc, and are called "arc" parameters. These parameters might be nuisance parameters, such as clock drift, or parameters of real geophysical importance, but which vary from day to day. The structure of the normal equations makes possible a fast matrix inversion.

In section 3 I describe another scheme for finding the solution to the normal equations which is a generalization of arc-parameter elimination. This technique can be used when there is an ordering of the normal equations so that most of the coupling (i.e., off diagonal elements) in the normal equations is between nearby parameters, with some coupling to global parameters. For example, in analyzing a single day of VLBI data, some parameters such as daily station position are on for the whole experiment, i.e., they are influenced by all of the data. These would be the global parameters. Other parameters, such as clocks, or atmospheres, are modeled as piecewise linear functions. The parameters which describe these functions are on for only a finite amount of time. Therefore they will couple with the global parameters,and other parameters whose "on-times" overlap.By squeezing out parameters as they are turned you can substantially reduce the time to solve the normal equations. I call this algorithm OMI, for Ordered Matrix Inversion.

In section 4 I briefly compare OMI with the B3D method of L. Petrov. I argue that OMI more general, and easier to implement.

Section 5 describes Fortran code which implements this algorithm, and tells where it is located.

2 Arc Parameter Elimination

In this section we describe the arc parameter elimination algortihm. This is suitable when we are dealing with normal equations which arise from combining many different experiments, and some of the parameters are common to all of the experiments, while others appear only in a single experiment. The combined normal equations can be written as.

$$N_{tot} A_{tot} = B_{tot} \tag{1}$$

Here N_{tot} is the total normal matrix, A_{tot} are the parameters we are solving for, and B_{tot} are the "O-C"'s. This can be re-written as:

$$\begin{pmatrix} N_{gg} & N_{g1} & N_{g2} & ... & N_{gn} \\ N_{1g} & N_{11} & 0 & ... & 0 \\ N_{2g} & 0 & N_{22} & ... & 0 \\ ... & ... & ... & ... & ... \\ N_{ng} & 0 & 0 & ... & N_{nn} \end{pmatrix} \begin{pmatrix} A_g \\ A_1 \\ A_2 \\ \\ A_3 \end{pmatrix} = \begin{pmatrix} B_g \\ B_1 \\ B_2 \\ \\ B_3 \end{pmatrix} \tag{2}$$

here the N's are rectangular sub-matrices. The N_{gg} are the normal equations restricted to the global parameters; the N_{ii} the normal equations restricted to the arc parameters for a given arc; and the N_{gi} the cross terms between the arc parameters and the global parameters. There are no cross terms between the different arcs. In general the size the sub-matrices are different for each arc. Because N is symmetric overall we have:

$$N_{gg}^T = N_{gg} \tag{3}$$

$$N_{ii}^T = N_{ii} \tag{4}$$

$$N_{gi}^T = N_{ig} \tag{5}$$

A straightforward algorithm for the solution vector, that is, the A's, takes on order of $\dim(N_{tot})^3/6$ multiplications, where $\dim(N_{tot})$ is the total number of parameters. If we are also interested in the covariance matrix, that is N_{tot}^{-1}, we need to do another $\dim(N_{tot})^3/3$ multiplications. For a standard

global solution the number of parameters is on the order of 10^6. A straightforward solution of either the solution vector or the covariance matrix would be almost impossible.

Apart from the first "row" and "column" of Eq. (2) the total normal matrix is strictly block diagonal. This allows a considerable speed up, both in the solution of the normal equation, and in the calculation of the covariance matrix.

2.1 Solution of Normal Vector

Equation (2) can be rewritten as the set of coupled equations:

$$N_{gg}A_g + \sum_i N_{gi}A_i = B_g \tag{6}$$

$$N_{ig}A_g + N_{ii}A_i = B_i \tag{7}$$

The second of these, Eq. (7) can be solved for the A_i

$$A_i = N_{ii}^{-1}B_i - N_{ii}^{-1}N_{ig}A_g \tag{8}$$

which when substituted into Eq. (6) gives

$$\left(N_{gg} - \sum_i N_{gi}N_{ii}^{-1}N_{ig}\right)A_g = B_g - \sum_i N_{gi}N_{ii}^{-1}B_i \tag{9}$$

which can be readily solved for the global parameters A_g:

$$A_g = \left(N_{gg} - \sum_i N_{gi}N_{ii}^{-1}N_{ig}\right)^{-1}\left(B_g - \sum_i N_{gi}N_{ii}^{-1}B_i\right) \tag{10}$$

The solution for A_g can in turn be substituted into equation (8) to find the solution for the arc parameters.

2.2 Solution of Covariance Matrix

In addition to the solution of the normal matrix, that is, the A's, we are also interested in the formal errors of the parameters, in which case we need the diagonal elements of N_{tot}^{-1}. We may also be interested in the correlation and covariance information, in which case we also need the off-diagonal elements. Consider the equation

$$\begin{pmatrix} N_{gg} & N_{g1} & N_{g2} & ... & N_{gn} \\ N_{1g} & N_{11} & 0 & ... & 0 \\ N_{2g} & 0 & N_{22} & ... & 0 \\ ... & ... & ... & ... & ... \\ N_{ng} & 0 & 0 & ... & N_{nn} \end{pmatrix} \begin{pmatrix} M_{gg} & M_{g1} & M_{g2} & ... & N_{gn} \\ M_{1g} & M_{11} & M_{12} & ... & M_{g1} \\ M_{2g} & M_{21} & M_{22} & ... & M_{g2} \\ ... & ... & ... & ... & ... \\ M_{ng} & M_{n1} & M_{n2} & ... & N_{nn} \end{pmatrix} = \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{pmatrix} \tag{11}$$

where $M_{tot} = N_{tot}^{-1}$. and each of the smaller M's is a rectangular submatrix. This leads to the coupled set of matrix equations:

$$N_{gg}M_{gg} + \sum_i N_{gi}M_{ig} = I \tag{12}$$

$$N_{ig}M_{gg} + N_{ii}M_{ig} = 0 \tag{13}$$

$$N_{ig}M_{gi} + N_{ii}M_{ii} = I \tag{14}$$

$$N_{ig}M_{gj} + N_{ii}M_{ij} = 0 \quad (i <> j) \tag{15}$$

Equation (13) can be re-written as:

$$M_{ig} = -N_{ii}^{-1} N_{ig} M_{gg} \tag{16}$$

which gives the covariance of the arc and global parameters. When substituted into equation (12) we get

$$\left(N_{gg} - \sum_i N_{gi} N_{ii}^{-1} N_{ig} \right) M_{gg} = I \tag{17}$$

or

$$M_{gg} = \left(N_{gg} - \sum_i N_{gi} N_{ii}^{-1} N_{ig} \right)^{-1} \tag{18}$$

Equation (18) gives the covariance matrix of the global parameters with each other, and the form is what you would naively expect based on Eq. (10). The square root of the diagonal elements are, of course, the formal errors for the global parameters. Substituting the transpose of equation (16) into equation (14) we get:

$$M_{ii} = N_{ii}^{-1} + N_{ii}^{-1} N_{ig} M_{gg} N_{gi} N_{ii}^{-1} \tag{19}$$

which is the covariance of the arc parameters for a particular arc with each other. Again, the square root of the diagonal elements are the formal errors of the arc parameters for this particular arc.

If all we are interested in is formal errors, or of the covariances of the arc parameters with themselves, the global parameters with themselves, or of the arc-global covariance, this is as far as we need to go. In fact, this is typically as far as solve goes. However, if we want to find the correlation of arc parameters in one arc with the arc parameters in another arc we need to go further. Transposing equation (16) and making the substitution $i \rightarrow j$, and then putting the result into equation (15) we find:

$$M_{ij} = N_{ii}^{-1} N_{ig} M_{gg} N_{gj} N_{jj}^{-1} \tag{20}$$

which gives the "arc-arc" covariance, i.e., the covariance of arc-parameters in one arc with those of arc-parameters in another arc.

2.3 Timing Considerations

In this subsection we compare the time for arc-parameter elimination with the straightforward solution. For simplicity we assume that all arcs have the same number of parameters. Let

$$\begin{aligned} N_{arc} &= \text{Number of arcs} && (21)\\ N_A &= \text{Parameters per arcs} && (22)\\ N_G &= \text{Global parameters} && (23)\\ N_T &= \text{Total parameters} && (24)\\ &= N_G + N_{arcs} N_A && (25) \end{aligned}$$

Then the naive, straightforward solution to the normal equations goes like

$$time\_naive = N_{tot}^3/2 \tag{26}$$

Time for Forward Solution. The matrix calculations we do in the forward direction, and the number of operations is summarized below:

Computation	Number of multiplications
N_{aa}^{-1}	$N_A^3/2$
$N_{ga}N_{aa}^{-1}$	$N_G N_A^2$
$(N_{ga}N_{aa}^{-1})\,N_{aa}$	$N_G^2 N_A/2$ (the factor of $1/2$ because the matrix is symmteric)
Total per arc	$N_A(N_A+N_G)^2/2$

(27)

At the end of the forward solution we do one final inversion. The total number of operations is then:

$$time\_forward = N_G^3/2 + N_{arc} \times N_A \times (N_A+N_G)^2/2 \quad (28)$$

$$= N_G^3/2 + (N_{tot}-N_G)(N_A+N_G)^2/2 \quad (29)$$

$$\simeq N_T\,(N_A+N_G)^2\,/2 + O(1) \quad (30)$$

where I have kept the leading term in N_T. The ratio of the naive time to the forward time is:

$$time\_naive/time\_forward = \left(\frac{N_T}{N_A+N_G}\right)^2 \quad (31)$$

$$\simeq N_{arc}^2\left(\frac{N_A}{N_A+N_G}\right)^2 \quad (32)$$

If we have a 10^3 arc solution, then arc-parameter elimination saves a factor of 10^6 in time, which is substantial indeed.

Time for Back Solution. Usually we are only interested in the covariance for the parameters of a particular arc. We are not interested in the full covariance matrix which involves arc-arc correlations. The table below summarizes the calculations needed, and the number of operations needed in the usual case.

Computation	Number of multiplications
$M_{gg}\,(N_{ga}N_{aa}^{-1})$	$N_G^2 N_A$
$(N_{aa}^{-1}N_{ag})\,M_{gg}\,(N_{ga}N_{aa}^{-1})$	$N_A^2 N_G/2$
Total per arc with storage	$N_A N_G(N_G+N_A/2)$

(33)

This table assumes that we store the matrix $N_{ga}N_{aa}^{-1}$ which is calculated in the forward direction. If we don't do so we need an additional $N_A^2(N_G+N_A/2)$ multiplications per arc, as can be see from the previous table. The total time for the back solution is:

$$time\_back = N_{arc}N_A N_G(N_G+N_A/2) \quad (34)$$

$$= (N_{tot}-N_G)\,N_G \times (N_G+N_A/2) \quad (35)$$

$$\simeq N_{tot} \times N_G \times (N_G+N_A/2) \quad (36)$$

The ratio of the naive solution time to the back solution time is:

$$time\_naive/time\_back = \frac{N_{tot}^2}{N_G(2N_G+N_A)} \quad (37)$$

$$\simeq N_{arc}^2\frac{N_A^2}{N_G(2N_G+N_A)} \quad (38)$$

which is about the same size as for the forward solution.

3 Solution of Normal Equations by Reduction and Augmentation

In this section we give a scheme for solving arbitrary normal equations which is closely related to the arc-parameter elimination of the last section. At the end of the section we specialize to the case of solve.

3.1 Solution of Normal Vector

We start by dividing the normal equaitons into two sets of parameters which we will label g and a. Without loss of generality the normal equation can then be written as:

$$\begin{pmatrix} N_{gg} & N_{ga} \\ N_{ag} & N_{aa} \end{pmatrix} \begin{pmatrix} A_g \\ A_a \end{pmatrix} = \begin{pmatrix} B_g \\ B_a \end{pmatrix} \tag{39}$$

This matrix equation can be rewritten as the two coupled equations

$$N_{gg}A_g + N_{ga}A_a = B_g \tag{40}$$
$$N_{ag}A_g + N_{aa}A_a = B_a \tag{41}$$

Equation (41) can be solved for A_a:

$$A_a = N_{aa}^{-1}B_a - N_{aa}^{-1}N_{ag}A_g \tag{42}$$

which when substituted into equation (40) leads to:

$$\left(N_{gg} - N_{ga}N_{aa}^{-1}N_{ag}\right)A_g = B_g - N_{ga}N_{aa}^{-1}B_a \tag{43}$$
$$N'_{gg}A_g = B_g - N_{ga}N_{aa}^{-1}B_a \tag{44}$$

This in turn can be solved for the A_g

$$A_g = N_{gg}^{'-1}\left(B_g - N_{ga}N_{aa}^{-1}B_a\right) \tag{45}$$

which can be substited into (42) to find the A_a.

The process of reducing the size of the normal equations is sometimes called "squeezing-out" the A_a. The reduced normal equations can again be reduced by another partition. We can continue doing so as long as we want. When we are finally done, we invert the last equation, and start building up the solution to the normal vector.

3.2 Derivation of Covariance Information

In this section we derive the equations for the covariance. Substitituting Eq. (45) into Eq. (42) we find

$$A_a = N_{aa}^{-1}B_a - N_{aa}^{-1}N_{ag}^{-1}N_{gg}^{'-1}\left(B_g - N_{ga}N_{aa}^{-1}B_a\right) \tag{46}$$

Combining this with Eq. (45) we arrive at the matrix equation:

$$\begin{pmatrix} A_g \\ A_a \end{pmatrix} = \begin{pmatrix} N_{gg}'^{-1} & N_{gg}'^{-1} N_{ga} N_{aa}^{-1} \\ N_{aa}^{-1} N_{ag} N_{gg}'^{-1} & N_{aa}^{-1} + N_{aa}^{-1} N_{ag} N_{gg}'^{-1} N_{ga} N_{aa}^{-1} \end{pmatrix} \begin{pmatrix} B_g \\ B_a \end{pmatrix} \tag{47}$$

$$= N_{tot}^{-1} \begin{pmatrix} B_g \\ B_a \end{pmatrix} \tag{48}$$

There are several comments to be made concerning this equation.

1. The covariance of the A_g is what you would niavely expect, namely $N_{gg}'^{-1} = (N_{gg} - N_{ga} N_{aa}^{-1} N_{ag}^{-1})^{-1}$.
2. The covariance of the A_a is $N_{aa}^{-1} + N_{aa}^{-1} N_{ag} N_{gg}'^{-1} N_{ga} N_{aa}^{-1}$.
3. More generally, given the covariance of the reduced normal equations (43), and the matrices N_{ga} and N_{aa}^{-1} it is straightforward to construct a solution to the normal equations one level up.

The general scheme is then clear. There are two processes: reduction and augmentation. In reduction we succesively squeeze out parameters until we are as far down as we can go. We solve for the remaining parameters. We then use this solution to solve for the "arc-parameters" one level up. This in turn is used to solve for the arc-parameters one level further up, and so on, until we are done. If we are only interested in the values of the arc-parameters, and not their formal errors, we are done. If we are interested in the formal errors, then we also need to build up the covariance matrix level by level.

3.3 Application to solve

The normal equations for an individual experiment in solve are sparse. The parameters typically consist of a few "arc-global" parameters, such as station position, EOP and baseline clocks, which take the same value for the entire arc. In addition there are "arc-arc" parameters where the normal equations are only influenced by a (usually small) subset of the total data. In contrast to the situation in arc-parameter-elimination, arc parameters which are adjacent to each other in time couple. For example, the clocks and the atmospheres are usually modeled as piecewise linear functions. If the atmosphere tabular points occur each hour, then each atmosphere partial is only effected by two hours of data–one on either side. Because of this there is no coupling with atmosphere or clock partials which are more than one hour apart. We can use the local nature of the coupling to derive a fast method of solving the normal equations.

The general scheme is as follows. We process the normal equations in time order, squeezing out parameters as they are turned off. At the end of the forward direction we are left with parameters which remain on at the end of the experiment. This includes the parameters which are on for the whole experiment, which is often all that we are interested. The solution to the reduced normal equations gives us the estimates for these parameters and their covariances. If this is all that we are interested in, then we are done. If we are interested in the values of the parameters which are squeezed out, we can do a simple back solution to find them. If we are interested in the covariance of these squeezed out parameters, we can do a more involved back solution. In the following we make this discussion concrete.

At any given stage in the processing, we divide the parameters into three sets. 1.) The "current-arc" parameters are the parameters we are getting ready to squueze out. 2) The "current-global" parameters are all parameters which couple to the current-arc parameters; 3) The remaining parameters do not couple to the current-arc parameters, although they may to the global parameters. We will label these three sets of parameters a, g, and r.

Forward Direction. We start by finding the first set of parameters which turn off.. Let the time they turn off be given by *tend*. These parameters will be the current-arc parameters. The current global parameters will be any of the other parmeters which were turned on prior to *tend.* The remaining parameters are those parameters which do not turn on until after *tend,* and hence do not couple to the current-arc parameters. By assumption, the normal equations take the form:

$$\begin{pmatrix} N_{gg} & N_{gr} & N_{ga} \\ N_{rg} & N_{rr} & 0 \\ N_{ag} & 0 & N_{aa} \end{pmatrix} \begin{pmatrix} A_g \\ A_r \\ A_a \end{pmatrix} = \begin{pmatrix} B_g \\ B_r \\ B_a \end{pmatrix} \tag{49}$$

We can solve this equation for the A_a:

$$A_a = N_{aa}^{-1} B_a - \left[N_{aa}^{-1} N_{ag} \right] A_g \tag{50}$$

After squeezing out the current-arc parameters, the reduced normal equations are:

$$\begin{pmatrix} N_{gg} - N_{ga} \left[N_{aa}^{-1} N_{ag} \right] & N_{gr} \\ N_{rg} & N_{rr} \end{pmatrix} \begin{pmatrix} A_g \\ A_r \end{pmatrix} = \begin{pmatrix} B_g - \left[N_{ga} N_{aa}^{-1} \right] B_a \\ B_r \end{pmatrix} \tag{51}$$

I have indicated by [] matrices which are related by transposition. For use in the backward direction we want to store the matrices and vectors N_{aa}^{-1}, $N_{aa}^{-1} B_a$, and $N_{ga} N_{aa}^{-1}$. These can be stored "in place":

$$N_{aa} \leftarrow N_{aa}^{-1} \tag{52}$$

$$B_a \leftarrow N_{aa}^{-1} B_a \tag{53}$$

$$N_{ga} \longleftarrow \left[N_{ga} N_{aa}^{-1} \right] \tag{54}$$

where $\leftarrow$ means replacement, and the order these equations appear in represents the order the calculations are done.

We now search for the next set of parameters which are turned off, and squeeze them out in turn.

The above form of the normal matrix made explicit the lack of coupling between the current-arc and remaining parameters. We do not need to rearrange the normal equations in this form in order to take advantage of this. The required matrix multiplications and inversions can be done by appropriately indexing into the original normal matrix or by copying the required matrix elements into temporary arrays, and doing the calculations upon them, and copying them back Although this seems like a lot of copying, since the number of current-arc and current-global parameters at any time is small, the size of the temporary arrays is small, and the overhead involved in doing the copying is also small.

Final Inversion. At the very end of the forward direction we invert the remaining matrix. This might happen when we have no more current-arc parameters, or when the matrix is small

enough that the extra book-keeping used in squeezing out parameters is not worth it. In any case, when the final matrix is inverted, we are left with a solution for the remaining parameters, and their covariances. For many purposes these are the only parameters we are interested, and we could stop at this stage.

Augmentation: Backward Direction. In augmentation we start with the reduced normal equations at some level, and derive the equations one level up. If all we are interested in is the solution vector, we can use Eq. (50) to find the arc parameters A_a we are adding. We have previously calculated and stored $N_{aa}^{-1}N_{ag}$ so this computation is very simple.

If we are also interested in the full covariance we need to do some extra work. Explicity, assume that covariance matrix at some stage is given by:

$$Cov(g,r) = \begin{pmatrix} M_{gg} & M_{gr} \\ M_{rg} & M_{rr} \end{pmatrix} \tag{55}$$

Then the covariance solution to the augmented normal equation is:

$$Cov(g,r,a = \begin{pmatrix} M_{gg} & M_{gr} & -M_{gg}N_{ga}N_{aa}^{-1} \\ M_{rg} & M_{rr} & -M_{gr}N_{ga}N_{aa}^{-1} \\ -N_{aa}^{-1}N_{ag}M_{gg} & -N_{aa}^{-1}N_{ag}M_{gr} & N_{aa}^{-1} + N_{aa}^{-1}N_{ag}M_{gg}N_{ag}N_{aa}^{-1} \end{pmatrix} \tag{56}$$

Note that in the lower-right corner of the covariance matrix there is no term involving $N_{aa}^{-1}N_{ar}M_{rr}N_{ar}N_{aa}^{-1}$ since by assumption N_{ar} vanishes. To find this augmented matrix we must calculate the following matrices: 1.) $M_{gg}\left(N_{ga}N_{aa}^{-1}\right)$; 2.) $M_{gr}\left(N_{ga}N_{aa}^{-1}\right)$; and 3.) $\left(N_{aa}^{-1}N_{ag}\right)\left(M_{gg}N_{ag}N_{aa}^{-1}\right)$ where I have enclosed by (..) matrix combinations which have been previously calculated.

The reason that this entire process is faster than the algorithm currently used in solve is that in both the squeezing out and augmentation phase we rely on submatrices of the normal or covariance matrices being 0. A random choice of what parameters to use as current-arc parameters will not in general lead to a lot of zeros. Fortunately, there is a natural way of choosing the parameters to squeeze out, namely the time ordering of the parameters.

It is clear that this algorithm will work for any situation where there is a natural ordering of the parameters. For example, this could be used in matrices which deal with finite difference models of bridges, where the ordering parameter would be distance along the bridge. For this reason I call the general technique Ordered Matrix Inversion, or OMI.

3.4 Timing Consideration

In this section we calculate the time involved to solve for a typical solution using the algorithm outlined above. Assume that there are N_G global parameters, that is parameters which are on for the whole experiment. For simplicity assume that the clocks and atmospheres are the only time-varying parameters, and that they both parametrized in the same way. For example, both have one hour rate breaks. Let N_{arc} be the number of tabular points for the clocks and atmospheres, and N_A the number of clocks plus atmospheres. Typically we have:

$$N_A = N_{clk} + N_{atm} = 2N_{stat} - 1$$

where N_{clk} are the number of clocks, N_{atm} the number of atmospheres, and N_{stat} the number of stations.The total number of parameters is:

$$N_T = N_G + N_{arc} \times N_A$$

For a typical 7 station experiment where we estimate EOP, gradients, and baseline clocks, and clocks and atmospheres have half-hour rate breaks, we have:

$$\begin{aligned} N_G &= 71 && (57)\\ N_A &= 13 && (58)\\ N_{arcs} &= 49 && (59)\\ N_T &= 708 && (60) \end{aligned}$$

The time it takes to find the solution using the standard solve algorithm is:

$$time\_solve = \dim(N_T)^3/2 \tag{61}$$

We want to compare this with a solution using the techniques of the previous section.

Forward Direction. For each arc in the forward direction we have the following steps involving matrix multiplication or inversion:

Computation	Number of multiplications
N_{aa}^{-1}	$n_a^3/2$
$N_{ga}N_{aa}^{-1}$	$n_g n_a^2$
$N_{ga}N_{aa}^{-1}N_{ag}$	$n_g^2 n_a/2$ (1/2 because the matrix is symmteric)
Total	$n_a(n_a+n_g)^2/2$

(62)

Here n_a and n_g are the dimensions of the current-arc and current-global matrices, and I have kept only these terms of order n^3. In the simplest case n_a and n_g will remain constant for each arc, and the total number of operations in the forward direction is then

$$N_{arcs} \times n_a(n_a+n_g)^2/2 \tag{63}$$

In the example at the start of this section, initially the current arc parameters will be the estimates of the tabular values for the first epoch of the clocks and atmospheres. The total number of tabular values per epoch is N_A. These current arc parametes couple to the truly global parameters. They also couple to the tabular points for the next epoch. Hence:

$$\begin{aligned} n_a &= N_A\\ n_g &= N_G + N_A \end{aligned}$$

Once the first set of arc parameters is squeezed out, the situation is exactly the same for the next set of arc parameters.

At the very end of the forward solution we invert the remaining global parameters. This has on order of $n_g^3/2$ multiplications. The total number of multiplications is then:

$$\begin{aligned} time\_forward &= \left[N_{arcs} \times n_a(n_a+n_g)^2 + n_g^3\right]/2\\ &= \left[(N_{tot}-N_G)\left(2N_A+N_G\right)^2 + \left(N_A+N_G\right)^3\right]/2\\ &\approx N_{tot}\left(2N_A+N_G\right)^2/2 \end{aligned}$$

where I have kept the leading terms in N_T. Dividing Eq. (61) by this we find:

$$time\_solve/time\_forward \simeq \left(\frac{N_T}{2N_A + N_G}\right)^2$$

Since N_T depends linearly on the number of arcs, that is, the number of rate breaks for the clocks and atmospheres, this expression will grow quadratically as this number is increased. For a typical experiment with 20-minute atmospheres, this ratio can easily be on the order of 100. For the concrete example given above the ratio is 40. If we are just interested in the solution vector and the covariance of the global parameters, we do not need to do a full back solution. The above ratio is the approximate speed up in this case.

Backward solution. In the backward solution we have the following operations:

Computation	Number of multiplications
$M_{gg}\left(N_{ga}N_{aa}^{-1}\right)$	$n_g^2 n_a$
$M_{rg}\left(N_{ga}N_{aa}^{-1}\right)$	$n_r n_g n_a$
$\left(N_{aa}^{-1}N_{ag}\right)M_{gg}\left(N_{ga}N_{aa}^{-1}\right)$	$n_a^2 n_g/2$
Total	$n_g n_a(n_g + n_r + n_a/2)$

(64)

In the simplest case, the number of current global and current arc parameters remains constaint, while the number of current other grows linearly. We return again to the example at the start of this section. At the beginning of the back solution, the current arc parameters will consist of the tabular values at $N_{arc} - 2$. The current-global parameters will be the truly global parameters, and the tabular values at $N_{arc} - 1$. The current remaining parameters will be the tabular values at N_{arc}. After the next round of augmentation, the current arc parameters will consist of the tabular values at $N_{arc} - 3$. The current-global parameters will be the truly global parameters, and the tabular values at $N_{arc} - 2$. The current remaining parameters will be the tabular values at $N_{arc} - 1$ and N_{arc}. Hence

$$n_{r1} = n_a \tag{65}$$
$$n_{r2} = 2n_a \tag{66}$$
$$n_{r3} = 3n_a \tag{67}$$
$$... \tag{68}$$

The total number of operations in the back direction will be:

$$\begin{aligned} num\_tot &= N_{arcs} \times n_g n_a(n_g + n_a/2 + (N_{arcs} - 1)/2 \times n_a) && (69)\\ &= N_{arcs} \times n_g n_a(n_g + n_a \times N_{arcs}/2) && (70)\\ &= (N_T - N_G) \times (N_G + N_A) \times [N_G + N_A + (N_T - N_G)] && (71)\\ &\approx (N_G + N_A) \times N_T^2/2 + O(N_T) && (72) \end{aligned}$$

where I have just kept the leading term in N_T. Dividing Eq. (61) by this we find:

$$time\_solve/time\_backward \simeq \frac{N_T}{N_A + N_G} \tag{73}$$

For the above example this will be a factor of 6. This grows approximately linearly with the number of arcs.

4 Comparison with B3D Algorithm of L. Petrov

In December of 1996 I became aware of work by L. Petrov on fast matrix inversion in solve. Petrov's fundamental insight was that the solve normal equations for each arc are sparse, while the matrix algorithm we used was for an arbitrary array. Petrov derived a technique for fast matrix inversion which he called the B3D algorithm. This method relies on the structure of the normal equations, in particular the fact that clocks and atmospheres are represented as piecewise linear functions. This scheme was subsequently extended to include piecewise linear EOP estimates. Petrov's has the restriction that the interval used for clocks, atmospheres and EOP must all be commensurate, and that the tabular points must all be aligned. By construction it makes use of the implicit structure of the normal equations used in solve: These paraticular parameters must occur in such and such an order. In fact Petrov's scheme actually expands the use of implicit data structures in solve, which makes modification of the code more difficult.

Because of the restrictions of Petrov's code, and also because I found his algebra hard to follow, I began to wonder if there was an alternative scheme for fast matrix inversion which took advantage of the sparseness of the normal equations, but did not have the restrictions of B3D, and also did not rely on the implicit structure of the normal equations. I also wanted a routine that I could understand. This is the genesis of the OMI algorithm derived in the previous section, and implemented in the following section in Fortran. I originally implemented this routine in March of 1997. This algorithm has the following advantages over B3D:

1. No implicit assumptions are made about the form of the normal equations. Instead the structure is derived from two auxiliary arrays *tb* and *te* which indiciate when the parameters are turned on. Hence this routine can be applied to an arbitrary time-tagged matrix.

2. There is no restriction on either the intervals or tabular points of clocks, atmosphers, and EOP.

3. If other parameters are estimated as piecewise linear functions, OMI will automatically handle them in an optimal fashion. In contrast, B3D would have to be rewritten to optimize performance.

4. OMI will automatically handle different parametrizations. For example, we could represent atmospheres as piecewise offsets, and this routine would handle this correctly. Or we could represent clocks as piecewise linear cubics.

5. OMI is close to being "plug-compatible" with the standard matrix inversion in solve. Since all of the bookkeeping is done internally, all you need to do is supply the auxiliary arrays *tb* and *te*. In contrast the implementation of B3D makes massive changes to solve to implement the book keeping required.

That being said, I believe that B3D may be slightly faster faster than OMI, say 30%. However, both of the algorithms result in a substantial speedup (on the order of 10-20 or greater) in the solution of the normal equations with a large number of stations. Suppose that B3D was twice as fast in matrix inversion as the algorithm described above, so that it speeed up matrix inversion by a factor of 40, instead of a factor of 20. Assume that matrix inversion takes 50% of the time of a solution in solve, and that a full solution takes 80 seconds. Then under B3D the time would be

reduced to 40+1 seconds, while using OMI the time would be reduced to 40+2 seconds. I think everyone would agree in this example that it implementing either OMI or B3D is worthwhile, since they both result in an almost 50% reduction in the time required to solve for the normal equations. However it is unclear wether the additional 1% reduction achieved by using B3D as opposed to OMI is worth the additional restrictions.

5 Fortran Code Which Implements Fast Matrix Inversion.

The following subroutine is the concrete implementation of the ideas mentioned in this note. The full routine involves a lot of bookkeeping, and is 600 lines of code. The fragment presented below just shows the bookkeeping involved. The use of the full routine in norml can be found by looking at the code in leo.gsfc.nasa.gov//data18/mk3/src/solve/norml_jmg_fast. It's use in arcpe is illustrated in leo.gsfc.nasa.gov//data18/mk3/src/solve/arcpe_jmg_fast.

On entry the array A is the normal matrix stored in lower triangular form, B is the "O-C" vector, and *tb* and *te* are arrays which contain the starting and stopping time of each parameter. *Nparm* is the number of parameters. The parameter *icov* controls whether we only want the covariance of the arc-global parameters (icov=0), the full covariance matrix (icov=2), or an approximation of the full covariance matrix. *(icov=1).*

The array *indb* is a key into the parameters sorted in increasing order of *tb,* and decreasing order of *te.* This is the order that we add the parameters to the active list. The array *inde* sorts the parameters by increasing *te* and decreasing *tb.* This is the order that we squeeze the parameters out. The routine that squeezes them out is called *reduce_normal.* When we are all done squeezing them out, we start the back solution. The routine that builds up the back solution is called *append_normal.*

```
C*******************************************************
      subroutine jmg_fast_solve(A,B,tb,te,nparm,icov)
      implicit none
C this routine inverts a normal matrix given in triangular form
C on entry A-- normal matrix in triangular form.
C          B-- O-C vector
C          tb, te  arrays which indicate how long a parameter is on for.
C on exit
C          A,B -- solution to normal equation.
C
C General scheme:
C  Solve normal equations iteratively, squeezing out stuff one as we go.
C  Start with smallest time interval, proceed to largest.
C
C For more detailed info, see memo by JMGipson, May 18, 1998.
C
      integer (kind=2) nparm
      double precision A(nparm*(nparm+1)/2)     !normal matrix
      double precision B(nparm)                 !normal vector
```

```
      double precision tb(nparm)                !parameter is turned on now.
      double precision te(nparm)                !  ...and off now.
      integer*2 icov                            !icov=0 don't compute covariance
                                                !icov=1 compute simplified
                                                !icov=2 Compute full

C local variables
C automatic local arrays used in sorting.
      integeR*2 indb(nparm)                   !sort key for start time.
      integer*2 inde(nparm)                   !sort key for end time
      integer*2 ib_ptr,ib_ptr0                !pointer into start time key
      integer*2 ie_ptr,ie_ptr0

C index arrays
      integeR*2 ind_tot(nparm)                !Index for all parameters this arc.
      integer*2 ind_glb(nparm)                !Index for the global parameters.
      INTEGER*2 ind_rst(nparm)                !Another index for the ''rest'' of the parms
      integer*2 num_rst

C number of parameters to add, number to subtract this arc, num globals
      INTEGER*2 max_arc,max_glb_per
      INTEGER*4 max_tot
      parameter (max_arc=200,max_glb_per=200)
      parameter (max_tot=max_arc*max_glb_per)

      INTEGER*2 ind_gta(max_tot)              !pointer into array containing global paramet
      INTEGER*4 igta_ptr

      INTEGER*2 num_arc,iarc                  !actual number of arcs, and do index
      integer*2 num_arc_arc,num_glb_arc
      integer*2 num_arc_vec(max_arc)          !number of arc parms this arc
      integer*2 num_glb_vec(max_arc)          !number of global this arc

      integer*2 num_tot                       !number of parameters (glb+arc)
      integer*2 num_2_add                     !number to add.

C Get two indices into parameters.
C Both indices sort on 1.) when they are turned on. and 2.) when they are off.
C   indb is sorted A.) First by increasing tb; B.) then by decreasing te;
C   inde is sorted A.) First by increasing te; B.) then by decreasing tb;
C This is done so that when we build up the ''global'' matrix we minimize reordering.
C

      call indexxr8id(nparm,tb,te,indb)
```

```
      call indexxr8id(nparm,te,tb,inde)

C start building normal equations in ''time order''

C ib_ptr -- points to where we are in putting in parameters.
C i.e. we have all observations before tb(indb(ib_ptr)).
C similarly for ie_ptr.
      ib_ptr=1
      ie_ptr=1
C where we are in the arcs.

      num_tot=0
      iarc=0
      igta_ptr=1

C------------------ this is the forward direction.---------------------
      FORWARD: DO
C----
C 1. Add in parameters
C    A.  Find out how many parameters to add to aglb.
C      This is # between tb(inbd(ib_ptr)) is before te(inde(ie))
        ib_ptr0=ib_ptr
        do while( ib_ptr .le. nparm .and. tb(indb(ib_ptr)) .le. te(inde(ie_ptr)))
           ib_ptr=ib_ptr+1
        end do
C   B.  augment the global matrix.
        num_2_add=ib_ptr-ib_ptr0
        ind_tot(num_tot+1:num_tot+num_2_add)=indb(ib_ptr0:ib_ptr-1)
        num_tot=num_tot+num_2_add

C if no more parameters, done with forward part.
        IF(ib_ptr .GT. nparm) exit
C
C 2. Squeeze out parameters
C   A. Find out which ones to remove
        ie_ptr0=ie_ptr
        do while(ie_ptr .le. nparm .and. te(inde(ie_ptr)) .le. tb(indb(ib_ptr)))
          ie_ptr=ie_ptr+1
        enddo
        num_arc_arc=ie_ptr-ie_ptr0

C At this stage inde(ie_ptr0:ie_ptr+num_arc_arc-1) contains index into arc parameters
C               ind_gta(igta_ptr:igta_ptr+num_glb-1)
C We also need the index into the global parameters.
        call get_other_ind(ind_tot,num_tot,inde(ie_ptr0),num_arc_arc,  ind_glb,num_glb_arc)
```

```
C
        call reduce_normal(a,b,nparm,inde(ie_ptr0),num_arc_arc,ind_glb,num_glb_arc)

C update arc stuff
        iarc=iarc+1
C need space for triangular matrix   Rectangular.
        num_arc_vec(iarc)=num_arc_arc
        num_glb_vec(iarc)=num_glb_arc
C this is index of globals this arc.
        num_tot=num_glb_arc
        ind_tot(1:num_tot)=ind_glb(1:num_tot)
C this is index of globals this arc.
        ind_gta(igta_ptr:igta_ptr+num_glb_Arc-1)=ind_glb(1:num_glb_arc)
        igta_ptr=igta_ptr+num_glb_arc
      END DO FORWARD

C done with forward direction.  Now invert remaining parameters.
      call indexed_invert(a,b,ind_tot,num_tot)

C**Start of back solution*****************************************************
C and add arc parameters an arc at a time.
      num_arc=iarc
      do iarc=num_arc,1,-1
         num_arc_arc=num_arc_vec(iarc)
         num_glb_arc=num_glb_vec(iarc)
         ie_ptr=ie_ptr-num_arc_arc
         igta_ptr=igta_ptr-num_glb_arc

         call get_other_ind(ind_tot,num_tot,ind_gta(igta_ptr),num_glb_arc,
     >         ind_rst,num_rst)

         call append_normal(a,b,nparm, inde(ie_ptr),num_arc_arc,
     >         ind_gta(igta_ptr),num_glb_arc,ind_rst,num_rst,icov)

C update index.
         ind_tot(num_tot+1:num_tot+num_arc_arc)=inde(ie_ptr:ie_ptr+num_arc_arc-1)
         num_tot=num_tot+num_arc_arc
      end do

      return
      end
C*******************************************************************************
```